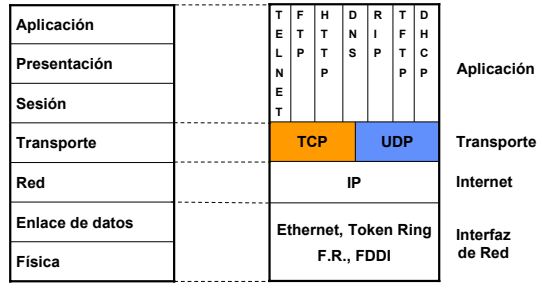
Tema 3:TCP/IP, UDP

- **Tema 3: Transporte: TCP/IP, UDP**
  - UDP
  - Funcionalidades de TCP
    - Control de errores
    - Control de flujo
    - Control de la congestión
  - Grafo de estados TCP
    - 3wHS
    - Finalización de la conexión.
  - Control de flujo
    - Sliding Window
    - TimeOuts: algoritmo de Van Jacobson para el temporizador TCP
  - Control de congestión
    - Slow Start
    - Congestion Avoidance
  - Sockets
    - Concepto de socket y llamadas al sistema en UNIX
    - Mapeo de llamadas a sockets con el grafo de estados de TCP
    - Servidores concurrentes e interactivos

Tema 3:TCP/IP, UDP

Modelo arquitectonico



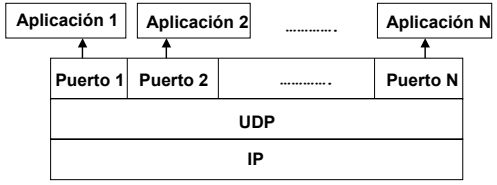
Tema 3:TCP/IP, UDP

- **Transporte:** nivel 4 cuyas funciones principales son:
  - Proporcionar conectividad extremo-a-extremo entre hosts
  - Sólo se incluyen los protocolos de transporte en los hosts (nunca en los routers a no ser que sean por alguna razón especifica, e.g. Gateways de aplicación)
  - Proporcionan estructuración de la información (segmentos TCP y datagramas UDP)
  - Multiplexación/demultiplexación de aplicaciones en transporte (concepto de puertos como identificador de las aplicaciones)
  - Uso de comunicaciones fiables y no fiables a nivel de transporte:
    - TCP: detección y control de errores y control de flujo y de la congestión extremo a extremo orientado a la conexión (para aplicaciones de datos) RFC 793
    - UDP: sólo detección de errores extremo a extremo **no** orientado a la conexión (para aplicaciones en tiempo real como son audio y vídeo) RFC 768

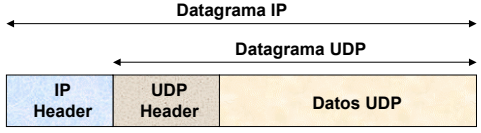
Tema 3:TCP/IP, UDP

- **UDP (User Datagram Protocol):**
  - Protocolo de transporte no-orientado a la conexión, cuya unidad de encapsulamiento es el datagrama UDP
  - Protocolo de datagrama de usuario
  - Ideal para comunicaciones en tiempo real
  - Cada escritura por parte de la aplicación provoca la creación de un Datagrama UDP
  - Cada datagrama UDP creado provoca la creación de un datagrama IP en el nivel 3
  - Si se pierde el datagrama IP o UDP es problema de la aplicación remota incorporar mecanismos de retransmisión.
  - Para IP, UDP es básicamente un interfaz de aplicación.
  - No añade fiabilidad.
  - Básicamnete es como un multiplexor/demultiplexor que sirve para enviar y recibir datagramas.
  - Puede transmitir más información en menor tiempo que TCP.

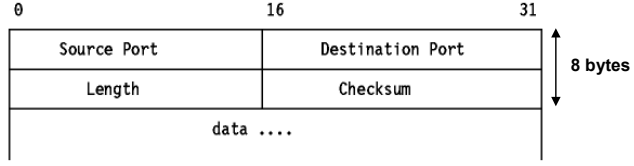
- **UDP (User Datagram Protocol):**
  - Foncionamiento del protocolo UDP



  - Datagrama IP con UDP



- **UDP (User Datagram Protocol):**
  - Datagrama UDP (8 bytes de cabecera)
    - **Puertos**: identifican a la aplicación origen y destino
    - **UDP length**: longitud total del datagrama UDP (campo redundante ya que IP lleva la longitud también)
    - **UDP checksum**: detector de errores que aplica a TODO el datagrama (recordar que checksum IP sólo cubría la cabecera IP)



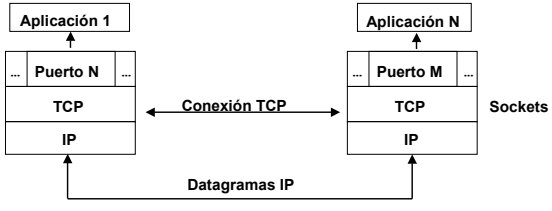
- **UDP (User Datagram Protocol):**
  - **UDP checksum**: detector de errores que aplica a TODO el datagrama (recordar que checksum IP sólo cubría la cabecera IP)
    - Para calcularlo necesitamos que la longitud del datagrama UDP sea un número par de octetos (para poder agruparlos en words de 16-bits)
    - Si no hay un número par entonces hacer padding (añadir un byte de 0s al final del datagrama)
    - El checksum además de la cabecera UDP cubre ciertos campos de la cabecera IP para hacer un “double-checking”. Esos campos son:
      - Direcciones IP, protocol field, total IP length
    - Para calcular el checksum, UDP crea una pseudo-cabecera con estos campos además de los campos del datagrama UDP
    - Si se detecta un error a nivel 4, el datagrama UDP se descarta y NO se genera ningún tipo de mensaje hacia el origen
    - **¿Porqué el checksum?** para detectar que NADIE ha modificado el datagrama UDP

- **UDP (User Datagram Protocol):**
  - **Maximum UDP size**:
    - En principio, como IP tiene una longitud máxima de 65335 bytes (16-bits de longitud de datagrama), la máxima longitud de un datagrama UDP sería 65335 – 20 bytes = 65315 bytes, lo que hace una longitud de datos de 65315 – 8 bytes = 65307 bytes
    - Pero **limitaciones del software**: casi todas las APIs (Aplication Program Interfaz) limitan la longitud de los datagramas UDP (lo mismo para TCP) a la máxima longitud que los buffers de lectura y escritura de los sockets (llamadas al sistema “read” y “write”)
    - Este límite es dependiente del SO: e.g: muchos de ellos (e.g.; FreeBSD) usan 8192 bytes como tamaño máximo del datagrama UDP

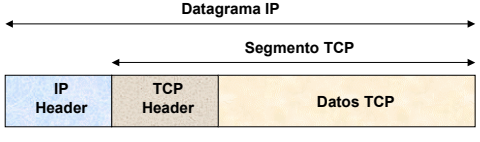
- **TCP (Transmission Control Protocol):**
  - Protocolo orientado a la conexión (estableciemiento de la conexión, envío de datos y cierre de la conexión).
  - Protocolo para el control de la transmisión.
  - Garantiza la entrega fiable a redes remotas.
  - Tal vez el protocolo **MAS complejo** e **importante** de la pila de protocolos. Unidad de datos es el "segmento TCP"
  - Proporciona fiabilidad mediante el control de flujo, control de errores y control de la congestión
  - Los segmentos pueden llegar fuera de orden (debajo hay IP que es no orientado a la conexión, o sea, datagrama), por tanto, TCP debe reordenar los segmentos antes de pasarlos a la aplicación

- **TCP (Transmission Control Protocol):**
  - **Características**
    - **Transferencia de datos** a través de un canal, TCP transfiere un flujo continuo de bytes a través de una red. TCP se encarga de trocear los datos de las aplicaciones y enviarlas a IP.
    - **Fiabilidad** TCP asigna un número de secuencia a cada byte transmitido y espera la respuesta afirmativa del TCP receptor (ACK).
    - **Control de flujo** con el que el TCP receptor avisa a l emisor mediante el envio de un ACK del número de bytes que aún puede recibir
    - **Multiplexación** mediante el uso de puertos como en UDP.
    - **Conexiones lógicas** es la unión del emisor con el receptor incluyendo los sockets.
    - **Full Duplex**

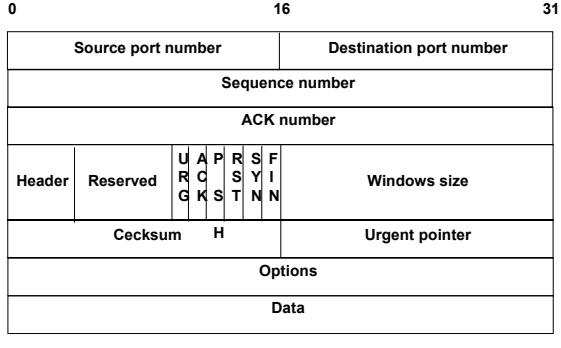
- **TCP (Transmission Control Protocol):**
  - Foncionamiento del protocolo TCP



  - Datagrama IP con TCP



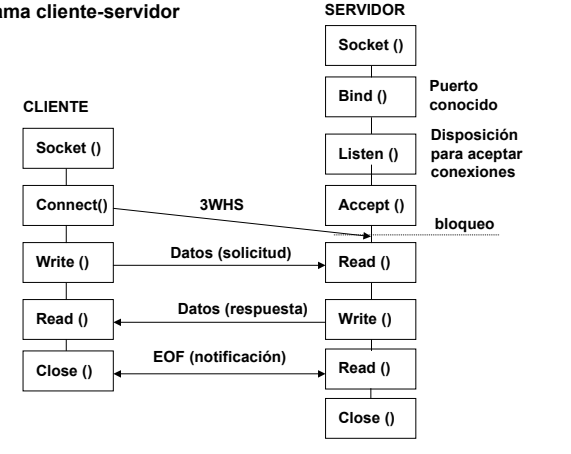
- **Cabecera TCP:**



- **Cabecera TCP:**
  - **Ports:** identifican las aplicaciones
  - **Sequence number:** identifica el primer byte de datos dentro de ese segmento de la secuencia de bytes enviados hasta ese momento.
    - ISN (Initial Seq. Number): primer número de secuencia escogida por el protocolo TCP
    - Seq. Number será un número a partir de ISN. Por tanto para saber cuantos bytes llevamos enviados hay que hacer *"Seq. Number – ISN"*
  - **Ack Number:** contiene el próximo número de seq. que el transmisor del ACK espera recibir
    - Por tanto es el "*Seq. Number+1"* del último byte recibido correctamente
    - Cuidado !!! TCP es FULL-DUPLEX: cada extremo mantiene un Seq. Number y un Ack Number
  - **Header length:** longitud total de la cabecera (opciones variable)

- **Flags:** hay 6 flags (bits) en la cabecera
  - **URG:** Urgent Pointer field valido
  - **ACK:** Ack Number es valido
  - **PSH:** el receptor debe pasar los datos a la aplicación tan rápido como sea posible
  - **RST:** "Reset" la conexión
  - **SYN:** Sincronización de los números de secuencia al iniciar la conexión
  - **FIN:** termina la conexión
- **Window Size**: tamaño de la ventana advertida por el receptor al transmisor (Sliding Window). Máxima ventana = 65535 bytes
- **Checksum**: de todo el segmento TCP (igual que UDP)
- **Urgent Pointer**: puntero al Seq. Number que indica la parte de datos urgentes dentro del campo de datos
- **Options**: opción de anunciar el MSS (Maximum Segment Size)

- **Programa cliente-servidor**

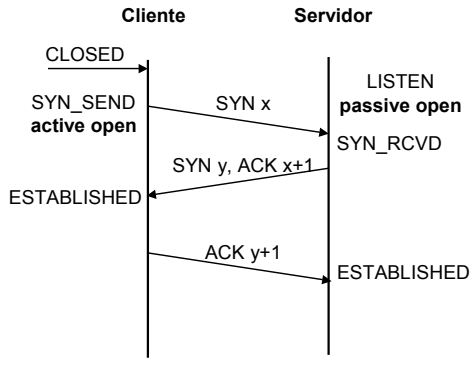


- **Establecimiento de la conexión TCP con 3WHS:**
  - Usa el 3-Way Handshake Algorithm:
    - El cliente envía un ***segmento SYN*** especificando el puerto destino del servidor, su puerto origen (escogido por el Kernel) y el ISN (Initial Seq Number) escogido al azar por el Kernel
    - El receptor (servidor) devuelve un ***segmento SYN+ACK*** reconociendo el segmento SYN e indicando su ISN
    - El cliente responde con un ***segmento ACK*** reconociendo el SYN+ACK
  - Una vez establecido la conexión se pasa a la fase de envío de datos
  - Es posible negociar **("indicar")** opciones (e.g.; indicar el MSS)
    - **MSS (Maximum Segment Size):** tamaño máximo de un segmento TCP.
      - Viene fijado por el Kernel: default = 536 bytes para un datagrama IP de 576 bytes (histórico X.25)
      - Sino, fijado por el MTU (Minimum Transfer Unit) menos cabeceras: e.g.; 1452 bytes en IEEE 802.3 (es 1500 – 8 LLC – 20 IP –20 TCP)

- **Estados TCP**

Active open	**CLOSED**	The socket is not being used.
	SYN_SENT	Actively trying to establish connection.
Passive open	**LISTEN**	Listening for incoming connections.
	SYN_RECEIVED	Initial synchronization of the connection under way.
	ESTABLISHED	Connection has been established.
Active close	**FIN_WAIT_1**	Socket closed; shutting down connection.
	CLOSING	Closed, then remote shutdown; awaiting acknowledgment.
	FIN_WAIT_2	Socket closed; waiting for shutdown from remote.
Passive close	**TIME_WAIT**	Wait after close for remote shutdown retransmission.
	CLOSE_WAIT	Remote shutdown; waiting for the socket to close.
	LAST_ACK	Remote shutdown, then closed; awaiting acknowledgment.

- **Establecimiento de la conexión TCP:**
  - Usa el 3-Way Handshake Algorithm:

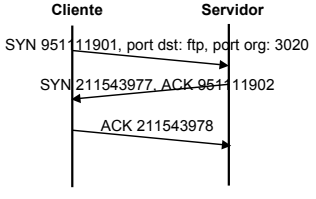


- **Ejemplo de establecimiento de la conexión TCP:**
  - Usa el 3-Way Handshake Algorithm:

11:27:13.771041 147.83.35.18.3020 > 147.83.32.14.ftp: **S 951111901:951111901(0)** win 32120 <mss 1460,sackOK,timestamp 86683767 0,nop,wscale 0> (DF)

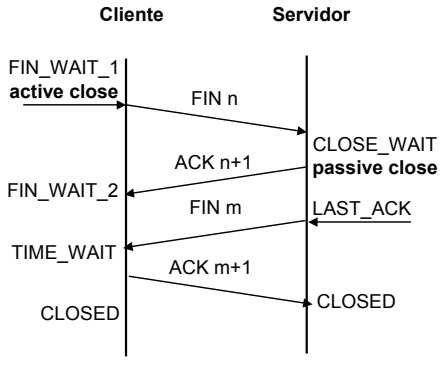
11:27:13.771491 147.83.32.14.ftp > 147.83.35.18.3020 : **S 211543977:211543977(0) ack 951111902** win 10136 <nop,nop,timestamp 104199850 86683767,nop,wscale 0,nop,nop,sackOK,mss 1460> (DF)

11:27:13.771517 147.83.35.18.3020 > 147.83.32.14.ftp: . 1:1(0) **ack 1** win 32120 <nop,nop,timestamp 86683767 104199850> (DF)



- **Cierre de la conexión TCP:**
  - El cierre de la conexión puede ser debido a varias causas:
    - El cliente o el servidor cierran la conexión (e.g.; LLS close())
    - Por alguna razón se envía un reset de la conexión (flag activo RST)
    - Cierre debido a una interrupción, e.g.; ^D o un ^C, etc, ...
  - El cierre normal es debido a un close del cliente lo que provoca el envío de 4 segmentos TCP
  - Como la conexión TCP es FDX cada dirección debe cerrar la conexión (envío de segmento FIN y de su correspondiente ACK)
  - Es posible que un extremo cierre su lado de la conexión y el otro no. En ese caso, el extremo que no ha cerrado puede enviar datos y el otro extremo los reconocerá (ACKs) aunque haya cerrado su conexión

- **Cierre de la conexión TCP:**



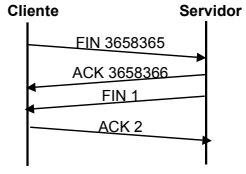
- Ejemplo de cierre de la conexión TCP

11:27:19.397349 147.83.32.14.ftp > 147.83.35.18.3020 : **F** 3658365:3658365(0) ack 1 win 10136 <nop,nop,timestamp 104200411 86684327> (DF)

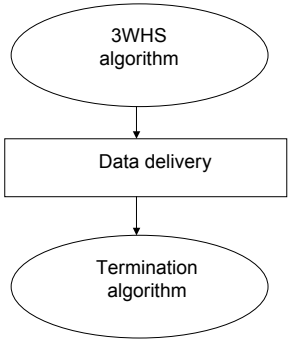
11:27:19.397370 147.83.35.18.3020 > 147.83.32.14.ftp: . 1:1(0) **ack 3658366** win 31856 <nop,nop,timestamp 86684330 104200411> (DF)

11:27:19.397453 147.83.35.18.3020 > 147.83.32.14.ftp: **F** 1:1(0) ack 3658366 win 31856 <nop,nop,timestamp 86684330 104200411> (DF)

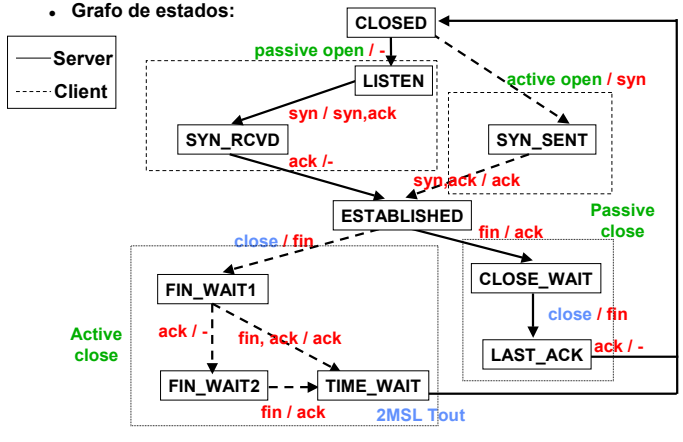
11:27:19.398437 147.83.32.14.ftp > 147.83.35.18.3020 : . 3658366:3658366(0) **ack 2** win 10136 <nop,nop,timestamp 104200412 86684330> (DF)



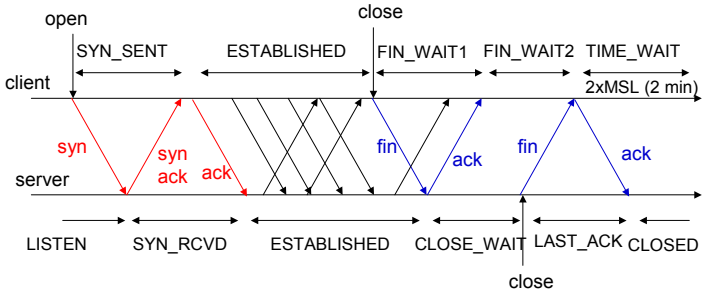
- **Grafo de estados en una conexión TCP**
  - En total 11 estados distintos: CLOSE, ESTABLISHED, LISTEN, LAST_ACK, ...



- **Grafo de estados:**



- **Grafo de estados:**



- **Grafo de estados:**
  - Comando **netstat**: permite listar las estructuras de datos de disversos protocolos de red (TCP, tablas de encaminamiento, tablas ARP, ...)

    netstat [-g | -m | -p | -s | -r | -f address_family] [-n] [-P protocol]

    g: multicast group

    m: shows streams statistics

    p: shows ARP table

    s: shows per-protocol statistics

    r: shows routing table

    P: statitistics of all socket associated to protocol

- **Ejemplo netstat**

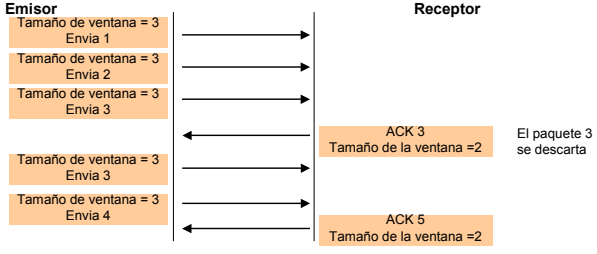
```
rogent:~>netstat -fn inet

UDP
   Local Address         Remote Address     State
-------------------- -------------------- -------
127.0.0.1.32889      127.0.0.1.123          Connected


TCP
   Local Address         Remote Address    Swind Send-Q Rwind Recv-Q  State
-------------------- -------------------- ----- ------ ----- ------ -------
147.83.31.7.22       147.83.35.18.1020     8688   8687 10136      0 ESTABLISHED
147.83.31.7.40336    147.83.31.7.32775    32768      0 32768      0 TIME_WAIT
147.83.31.7.2003     147.83.33.6.52649    32768      0  8760      0 TIME_WAIT
147.83.31.7.40343    147.83.31.7.32775    32768      0 32768      0 TIME_WAIT
147.83.31.7.40359    147.83.31.7.32775    32768      0 32768      0 TIME_WAIT
147.83.31.7.512      147.83.35.110.1183    8687      0  8760      0 CLOSE_WAIT
147.83.31.7.39913    147.83.35.14.6000     7840      0  8760     32 ESTABLISHED
147.83.31.7.40361    147.83.35.106.19     17423      0  8760    512 ESTABLISHED
147.83.31.7.957      147.83.33.10.2049     8760   2483  8760      0 ESTABLISHED
```

bytes in Tx buffer pendientes de ACK

bytes in Rx buffer pendientes de ser consumidos aplicación

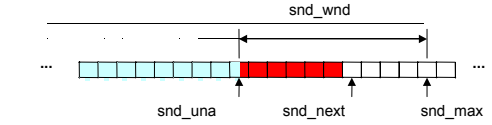
- **Tipos de aplicaciones que usan TCP:**
  - **Interactive aplications**: aquellas que envían poca cantidad de datos
    - Rlogin, telnet
    - Transmiten segmentos de tamaño muy pequeño (90 % de los segmentos tienen menos de 10 bytes de datos)
    - No es importante el control de la congestión
    - Importante los algoritmos de Nagle y los “Delayed ACKs”
  - **Bulk Transfer**: aplicaciones que envían gran cantidad de datos
    - FTP
    - Los segmentos llevan mas de 512 bytes de datos
    - Importante los algoritmos de control de la congestión (Slow Start, Congestion Avoidance, Fast Retransmit y Fast Recovery)
  - Que en las aplicaciones interactivas no sea importante el control de la congestión, no significa que no lo implementen, sólo que no se ven afectadas por este control de la congestión

- **Control de flujo (Sliding Window)**
  - El TCP receptor devuelve una ventana al TCP emisor.
  - Se especifica el número de octetos que el receptor esta preparado para recibir y el número de referencia por donde empieza.
  - Permite múltiples tramas no confirmadas (sin ACK)
  - Hay un límite para el número de tramas no confirmadas o pendientes, llamado ventana (*window)*



- **Control de flujo**

TCP implementa un protocolo de control de flujo mediante una ventana de tamaño deslizante.

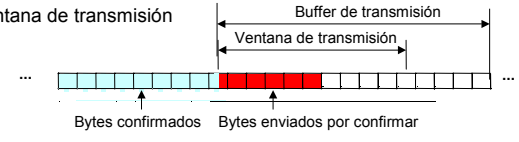


Leyend:

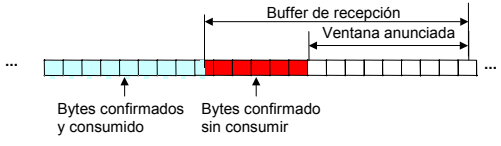
	Sent and acknowledged segment.
	Sent but unacknowledged segment.
	Unsent segment.
snd_wnd	advertised window.
snd_una	first unacknowledged segment.
snd_next	next segment to be sent.
snd_max	last segment that can be sent.

- **Control de flujo**

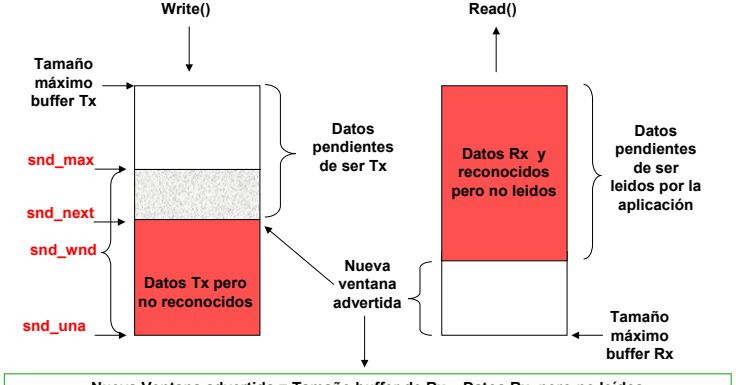
Ventana de transmisión



Ventana de recepción



- **Control de flujo en TCP**



- **Control de flujo**
  - El transmisor guarda en el buffer:
    - Los bytes enviados y no reconocidos por si ha de enviarlos otra vez
    - Los bytes que no se han transmitido
  - El receptor guarda en el buffer:
    - Los bytes recibidos y pendientes de consumir por la aplicación
  - Casos que se pueden dar:
    - Si el receptor es más rápido que el emisor, en este caso estara esperando contiamente datos y al ventana podría llegar a ocupar todo el buffer.
    - El receptor es igual de rápido que el emisor, en este caso se van transmitiendo datos y el bufer del receptor tiene datos no consumidos y datos que se han recibido del emisor.
    - El receptor es más lento que el emisor, en este caso la ventana advertida se ira reduciendo hasta que bloquee el emisor ya que no podra enviar más datos.

- **Control de errores**
  - Si un segmento se pierde y el emisor no recibe su ACK, los siguientes segmentos se van enviando y el receptor los reconocera con el ACK del anterior al que no se ha cecibido, pero el emisor necesita el ACK del segmento perdido. En el emisor cuando tengamos el timeout de dicho segmento lo retransmitira generando en el receptor un ACK del último paquete que llego.
  - Si un segmento llega al receptor, pero no se recibe su ACK en el emisor, pero si el ACK del siguiente segmento, la recepcion del mismo implica la recepción de todos los anteriores
  - El problema es como fijar los valores de time-out, que esta diseñado por los progamadores, normalmente se emplea el algoritmo de Van Jcobson en el que se utiliza varios valores promediados de los tiempos de en que tarde en enviar un segmento y recibir su ACK.

- **Control de errores**:
  - Computación del Temporizador de retransmisiones (**Algoritmo de Van Jacobson**):
    - Definimos:
      - RTT (Round Trip Time) como el tiempo de ciclo
      - RTO (Retransmission Time Out): temporizador de retransmisiones
    - El TCP transmisor mide la media del RTT (srtt) y la desviación media (rttvar)
    - El Time out se calcula como: RTO = srtt + 4 rttvar
    - El RTO se duplica cada vez que **se retransmite** un segmento (exponential backoff) → RTO = 2*RTO
  - Granularidad en TCP:
    - RTO se calcula en terminos de “slow-timer tics” (cada 500ms)
  - Medidas del RTT:
    - En función de los tics (aproximación dura)
    - Usando timestamps (opción del segmento TCP)

- **Control de errores:**
  - Computación del Temporizador de retransmisiones (Algoritmo de Van Jacobson + Algoritmo de Karn):
    - srtt: estimación de la media del tiempo de ciclo
    - rttvar: desviaciόm media de la medida del tiempo de ciclo
    - RTO: Temporizador de transmisión (Van Jacobson)
    - RTT_{medido}: medida del tiempo de ciclo (sólo cuando el ACK pertenece a un segmento NO retransmitido) (Karn)

$err = RTT_{medido} - srtt$

$srtt = srtt + g * err$ → **g = 1/8**

$rttvar = rttvar + h*(| err | - rttvar)$ → **h = 1/4**

RTO = srtt + 4 rttvar **no retransmisión**

RTO = 2 RTO **si retransmisión (Exponencial backoff)**

- **Datos interactivos:**
  - **Delayed ACKs:** se envían ACKs cada 2 segmentos recibidos o al cabo de un temporizador de 200 ms, esto es útil cuando se intenta agregar un máximo de bytes confirmados.

```
...
11:27:13.798849 147.83.32.14.ftp > 147.83.35.18.3020: P 9641:11089(1448)
                ack 1 win 10136 (DF)
11:27:13.800174 147.83.32.14.ftp > 147.83.35.18.3020: P 11089:12537
                (1448) ack 1 win 10136 (DF)
11:27:13.800191 147.83.35.18.3020 > 147.83.32.14.ftp: . 1:1(0) ack 12537
                win 31856 (DF)
11:27:13.801405 147.83.32.14.ftp > 147.83.35.18.3020: P 12537:13985
                (1448) ack 1 win 10136 (DF)
11:27:13.802771 147.83.32.14.ftp > 147.83.35.18.3020: P 13985:15433
                (1448) ack 1 win 10136 (DF)
11:27:13.802788 147.83.35.18.3020 > 147.83.32.14.ftp: . 1:1(0) ack 15433
                win 31856 (DF)
...
```

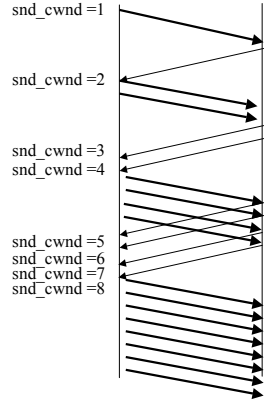

- **Control de congestión (Bulk Data Transfer):**
  - Control de la congestión debido a que enviamos muchos datos de golpey los buffers pueden desbordarse ya sea el de los host o los de los routers.
  - Mecanismo de control de la congestión: Slow Start, Congestion Avoidance, Fast Retransmit, Fast Recovery
  - Implementaciones TCP: muy variadas
    - Todas están obligadas a implementar Slow Start y Congestion Avoid.
    - TCP Tahoe (1988): slow start, congestion avoidance, Fast Retransmit
    - TCP Reno (1990): Tahoe + Fast recovery + TCP header prediction
    - TCP Sack: Reno+ Selective ACK
    - Otras: multicasting, routing tables, ...

- **Slow Start (SS) y Congestion Avoidance (CA):**
  - Funcionan conjuntamente
  - **Slow start** (Funcionamiento):
    - Se quiere efectuar una transmisión a la velocidad en la que podamos transmitir los datos sin perdidas.
    - Inyectar segmentos en la red a la velocidad a la que recibimos los ACKs desde el otro extremo
    - Ventana advertida (snd_wnd): ventana advertida por el receptor
      - ***Es un control de flujo impuesto por el receptor que es el que fija el valor de la ventana advertida***
    - Ventana de congestión (snd_cwnd): ventana que se inicializa a 1
      - ***Es un control de flujo impuesto por el transmisor***
    - Cada vez que se recibe un ACK → snd_cwnd (se incrementa en 1)
    - El incremento de segmentos transmitidos es exponencial

El transmisor transmite:

Ventana de TX = min (snd_wnd, snd_cwnd)

- **Slow Start (SS):**



Inicialmente { snd_wnd =10 ; snd_cwnd =1 }

Min (snd_wnd, snd_cwnd)

El incremento es muy rápido

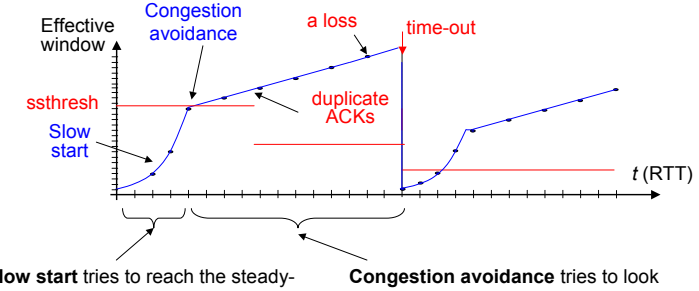
- **Slow Start y Congestion Avoidance:**
  - **Congestion Avoidance** (CA):
    - Si comenzamos con Slow Start, enviamos segmentos de forma exponencial, aumentando la ventana de congestión hasta que llenamos el buffer de algún router y se producen perdidas.
    - Si la red se congestiona queremos que la red se recupere (congestion avoidance) sin dejar de transmitir y comenzar de nuevo en un slow start
    - ***Es un control de flujo impuesto por el transmisor***
    - Congestion avoidance hace que cuando se detecta una perdida, en vez de transmitir exponencialemente segmentos, pasemos a transmitir de forma lineal hasta que se recupere la red.
    - Con Congestion Avoidance intentamos mantenernos en el límite de la transmisión de datos sin tener perdidas.

- **Congestion Avoidance (CA)**:
  - Define un nuevo parámetro, threshold (umbral) (**ssthresh**) que se inicializa a 65535 bytes
  - Entoces tenemos **snd_cwnd = 1** y **ssthresh = ventana máxima**
    - Si hay congestión:
      - Debido a una perdida (Tout) o a la recepción de ACKs duplicados → el threshold pasa a valer la mitad de la ventana de transmisión pero no por debajo de 2 segmentos, es decir:
        - **ssthresh=max[2, 1/2 min(snd_wnd, snd_cwnd)]**
      - Si además la congestión es por el salto del temporizador, **snd_cwnd=1**
    - Si **snd_cwnd < ssthresh** → *estamos en Slow Start (SS)*
      - Cada vez que recibamos un ACK, incrementamos la ventana de congestión de la siguiente manera
        - Si estamos en SS → snd_cwnd
    - Si **snd_cwnd ≥ ssthresh** → *estamos en Congestion Avoidance (CA)*
      - Cada vez que recibamos un ACK, incrementamos la ventana de congestión de la siguiente manera
        - Si estamos en CA → snd_cwnd = snd_cwnd + 1/ snd_cwnd

- **Slow Start y Congestion Avoidance:**

```
Initialization:
    snd_cwnd = 1 ;
    ssthresh = 65535 bytes;
Upon ack reception:
    if(snd_cwnd < ssthresh)      /* Slow Start */
        snd_cwnd = snd_cwnd + 1 ;
    else                         /* Congestion Avoidance*/
        snd_cwnd = snd_cwnd + 1/snd_cwnd;
When a time-out occurs:
    send snd_una ;
    snd_cwnd = 1 ;
    ssthresh = max(2, effective window / 2) ;
```


Slow Start y Congestion Avoidance:



Slow start tries to reach the steady-state operation point.

Congestion avoidance tries to look for higher available transmission rate.

- Detalles importantes
  - Que pasa si un cliente o un servidor no envían datos
    - El otro extremo no puede envíar ACKs (que pasaría si la ventana advertida valiera 0?). Se envían segmentos de prueba **("window proves")** de 1 byte cada cierto tiempo **("persist timer" acotado entre 5 y 60 sec)** para que el otro extremo pueda enviar ACKs advirtiendo una nueva ventana
    - Qué ocurre si un extremo deja de funcionar (crash). **"Keepalive timer"** se encarga de enviar cada 2 horas un segmento de prueba. Sino recibe respuesta, envía 10 pruebas cada 75 segundos y hace un reset

- **Sockets:**
  - Los sockets son el interfaz de comunicaciones en la red.
  - Un socket es una API (Aplication Program Interface) con los protocolos de comunicaciones como TCP/IP. Estas API fueron definidas por primera vez por la Berkeley Software Distribution (BSD).
  - Un socket se asocia a un puerto y a una dirección IP.
  - Para establecer una conexión mediante sockets hay que tener en cuenta dos caracteristicas:
    - La familia de sockets que se esta utilizando, que la más comun es la familia de protocolos internos de Unix y la familia de protocolos de Internet comoTCP y UDP.
    - El tipo de conexión que puede ser de circuito virtual orientado a conexión o de tipo datagrama no orientado a conexión.
  - Estructura de un socket

socket (familia, tipo, protocolo)

- **Sockets:**
  - Modelo cliente-servidor
    - Los servidores están esperando peticiones de los clientes
    - Cuando un cliente quiere un servicio (aplicación) debe establecer una conexión con el servidor. A la conexión se le llama en UNIX "socket"
    - Socket servidor, tiene la IP del servidor y el puerto escogido y conocida por la aplicación
    - Socket cliente, tiene la IP del cliente y un puerto efimero escogido por la aplicación de forma dinámica.
    - Un socket es un descriptor de fichero que apunta a una estructura de datos que representa la conexión de transporte
      - Sockets TCP (SOCKET_STREAM)
      - Sockets UDP (SOCKET_DGRAM)

- **Estructuras de datos en IP:**
  - Estructura dentro del SO (UNIX) que guarda la información relacionada con la familia (AF_INET, AF_INET6, AF_ROUTE, AF_UNIX) de protocolos

```
struct sockaddr_in {
    short     sin_family;     /* AF_INET para IPv4 */
    u_short sin_port;         /* 16 bits para puerto */
    struct    in_addr sin_addr;   /* 32 bits para IP */
    char      sin_zero[8];    /* 8 bytes no usados */
    }
struct in_addr {
    u_long  s_addr;           /* 32 bits para IP */
    }
```

- **Estructuras de datos en IP:**
  - Orden de los bits:
    - En redes siempre se guardan los octetos en formato **big-endian** (llamado formato de red)
    - En los ordenadores la información se puede guardar en big-endian o en little-endian

Puerto 80 = 0x0050

byte	Little-endian	Big-endian
n	50	00
n+1	00	50

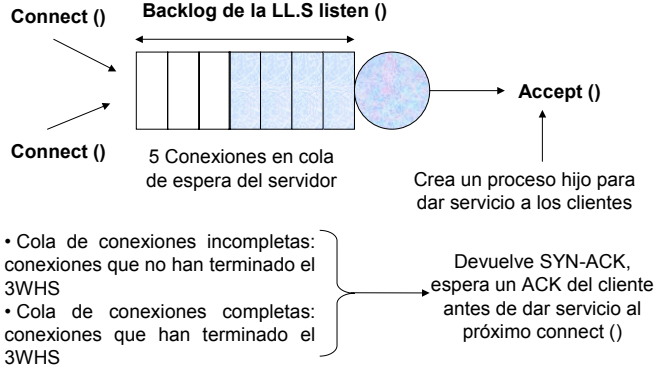
htons() → Host to Network short (16 bits)

ntohs() → Network to Host short (16 bits)

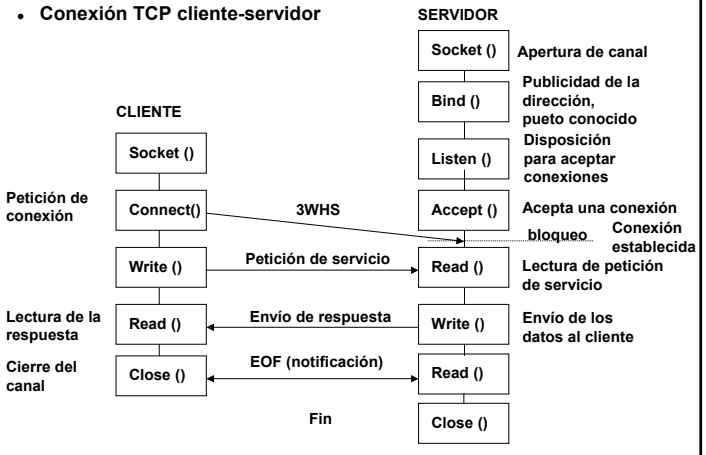
- **Puertos TCP/UDP**
  - Un puerto identifica la aplicación de red que deseamos utilizar
    - Puertos conocidos ("Well-known ports"): puertos que identifican la aplicación servidor (rango 1-1023), ver /etc/services. Servicios no estándares usan ports > 5000
      - 20/tcp ftp-data
      - 21/tcp ftp
      - 23/tcp telnet
      - 25/tcp smtp
      - 69/udp tftp
      - 80/tcp http
      - .....
    - Puertos efímeros ("ephimeral ports"): puertos usados por los clientes (típicamente entre 1024 y 5000) para BSD.

- **Sockets UNIX (Llamadas al sistema):**
  - **sfd=socket (af, type,protocol):** af define familia (UNIX, IPv4, IPv6), tipo (datagrama UDP, segmento TCP, datagrama IP, trama), protocol (por si la pila es multiprotocolo), devuelve un descriptor de fichero de tipo socket
  - **bind (sfd, *addr, addrlen):** hace que sfd apunte a la estructura de datos apuntada por addr (es la dirección IP destino, la origen ya la sabe)
  - **listen (sfd, backlog):** backlog es la cola asociada al socket servidor que indica cuantas conexiones pueden estar en 3WHS (en el estado SYN_RECV o pendientes de accept)
  - **connect(sfd, *addr, addrlen):** permite a un proceso cliente inicializar la comunicación (3WHS)
  - **accept (sfd, *addr, addrlen):** permite a un servidor aceptar una petición de un cliente
  - **close (sfd):** cierra un socket

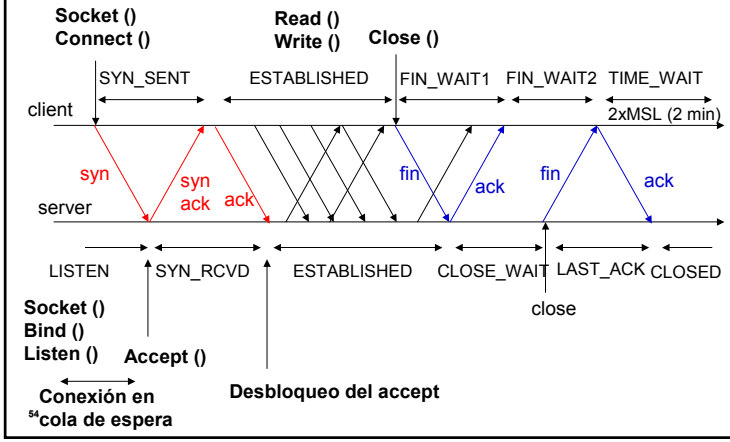
- **Relación entre las LL.S y el diagrama de estados TCP**



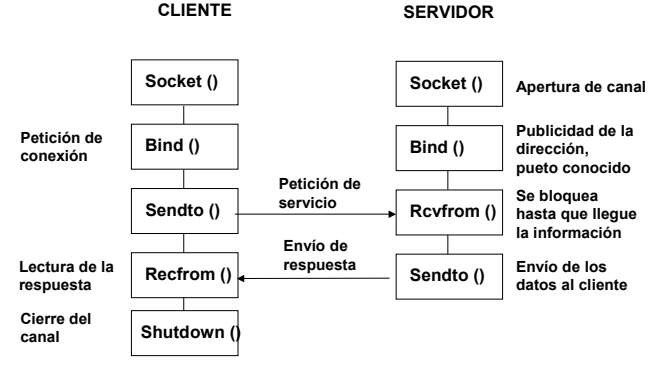
- **Conexión TCP cliente-servidor**



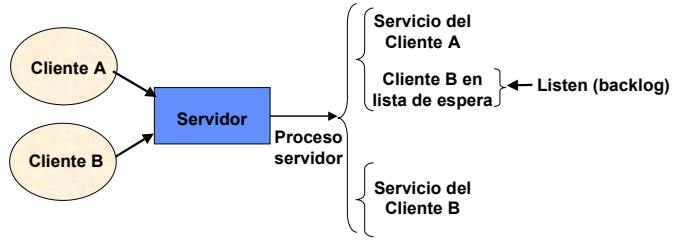
- Relación entre las LL.S y el diagrama de estados TCP



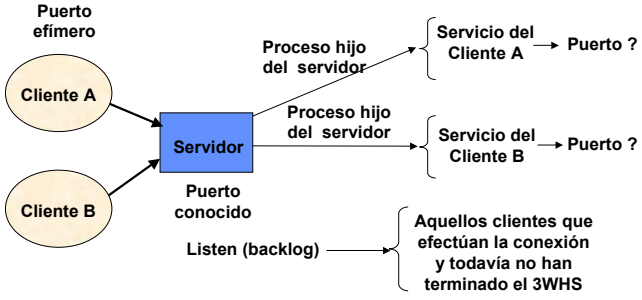
- **Conexión UDP cliente-servidor**



- **Tipos de servidores:**
  - Servidores interactivos: el servidor recoge una petición de servicio y la efectua inmediatamente. Es útil cuando el servicio no es muy complicado puesto que si recibe peticiones cuando esta ejecutando una origina retardos.



- **Tipos de servidores:**
  - Servidores concurrentes: Cuando el servidor recibe una petición de servicio de varios clientes a la vez y crea un nuevo proceso , hijo, para atenderla, así el servidor puede ir recibiendo peticiones de más clientes.



- **Opciones socket**
  - Se pueden obtener y modificar en UNIX con los comandos *getsockopt()* y *setsockopt()*

SOL_SOCKET:		
	SO_RCVBUF →	**Receiv. buffer size**
	SO_SNDBUF →	**Send buffer size**
IPPROTO_IP:		
	IP_OPT →	**IP Header Options**
	IP_TTL →	**Time To Live**
IPPROTO_TCP:		
	TCP_NODELAY →	**Disable Nagle algorithm**
	TCP_MAXSEG →	**Maximum Segment Size**
	TCP_KEEPALIVE →	**seconds of keepalive timer**

- **Netstat y el grafo de estados**
  - *netstat –a –f inet* → tabla con el estado de las conexiones en curso
    - **Proto**: protocolo TCP/UDP
    - **RecvQ/SendQ:** cola de recepción/envío
    - **Local Addr/Foreign Addr /STATE**
      - **Loc.Addr=***.p en estado LISTEN indica que un servidor está escuchando por el puerto "p" en todas las IP. **For.Addr=*.*** en estado LISTEN indica que un servidor espera conexiones desde cualquier IP y cualquier "puerto"
      - **Loc.Addr=IP_a.p_a** y **For.Addr=IP_b.p_b** en estado ESTABLISHED indica que se ha establecido una conexión al servidor con dirección IP_a con puerto p_a desde el cliente remoto IP_b con puerto p_b

- ***netstat –a –f inet*** → tabla con el estado de las conexiones en curso

Active Internet connections (servers and established)

Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State
tcp	0	0	aucanada.ac.u:xiostatus	teix.ac.upc.es:6000	TIME_WAIT
tcp	0	0	aucanada.a:wrs_registry	teix.ac.upc.es:6000	TIME_WAIT
tcp	0	0	aucanada.a:3com-webview	teix.ac.upc.es:6000	TIME_WAIT
tcp	0	0	aucanada:norton-lambert	teix.ac.upc.es:6000	TIME_WAIT
tcp	0	0	aucanada.ac.upc:ideesrv	teix.ac.upc.es:6000	TIME_WAIT
tcp	0	4	aucanada.ac.upc.:telnet	rogent.ac.upc.es:59463	ESTABLISHED
tcp	0	0	aucanada.ac.upc.es:6000	fonoll.ac.upc.es:44676	ESTABLISHED
tcp	0	0	aucanada.ac.upc.es:1023	fonoll.ac.upc.es:login	ESTABLISHED
tcp	0	0	*:ace-proxy	*:*	LISTEN
tcp	0	0	*:6000	*:*	LISTEN
tcp	0	0	*:netbios-ssn	*:*	LISTEN
tcp	0	0	*:www-http	*:*	LISTEN
tcp	0	0	*:auth	*:*	LISTEN